Arrow-bot: A Teaching Tool for Real-Time Embedded System Course

Mohamad Fauzi Zakaria^{1,a}

¹*Advanced Mechatronics Research Group (ADMIRE), Department of Mechatronics and Robotics Engineering, Faculty of Electrical and Electronic Engineering, Universiti Tun Hussein Onn Malaysia, 86400 Batu Pahat, Johor, Malaysia*

Abstract. This paper presents the design of a line following Arduino-based mobile robot for Real-Time Embedded System course at Universiti Tun Hussein Onn Malaysia. The real-time system (RTS) concept was implementing is based on rate monotonic scheduling (RMS) on an ATmega328P microcontroller. Three infrared line sensors were used as input for controlling two direct current (DC) motors. A RTS software was programmed in Arduino IDE which relied on a real-time operating system (RTOS) of ChibiOS/RT library. Three independent tasks of software functions were created for testing real-time scheduling capability and the result of temporal scope was collected. The microcontroller succeeded to handle multiple tasks without missed their dateline. This implementation of the RTOS in embedded system for mobile robotics system is hoped to increase students understanding and learning capability.

1 Introduction

Real-time embedded systems can be found in many technology systems such as smartphones, home appliances, terrestrial transport systems, factory machinery control and bank transaction systems [1]. Each of them should produce the right output before the time deadline to maintain the system performance and safety issues [2]. A knowledge and skill for designing and implementing a real-time system concept are essential needed by mechatronics engineer to produce the robust and reliable product.

Faculty of Electrical and Electronic Engineering in Universiti Tun Hussein Onn Malaysia (UTHM) offers a real-time embedded system course every semester to undergraduate mechatronics students. The goal of this course is to establish a comprehensive understanding of the concept, technique, and analysis of a real-time embedded system. A real-life implementation of the real-time system should be shown to students which aimed to increase their motivation and learning capability. Therefore, a mobile robot was selected to be a teaching tool for exposing to the student on how to design and implement real-time system concept in embedded microcontroller system.

There is one research paper was successfully implemented the RMS for an autonomous robot by Robert George and Yutaka Kanayama [3]. However, this paper only focused on a simple testing analysis of RMS and implemented on big size (60 x 60 x 70 cm) and weighted 60 kg of a UNIX-based mobile robot. Hence, it is not suitable for teaching and learning tool. Currently, there are a lot of research and commercial product of educational mobile robot with desktop size such as Khepera [4], e-Puck [5], and Lego Mindstorms

EV3 [6]. However, these robots are not using the popular open system. Therefore, it is difficult to students for learning microcontroller by themselves due to less community support and reference book. These days, the popular and low-cost open system is an Arduino-based microcontroller. There are a few commercial Arduino-based robots available in the market have been used as a teaching platform such as Zumo, Arduino-robot, m-Bot robot kit, and miniQ 2WD mobile robot. However, their complete design is complex due to more functions and need extra time for students to get used to them. So, in my opinion, it is the best approach if we could design a robotics system with minimum electronic components and compact in size for line following application. In consequence, it will reduce a lot of system learning time and less difficult to do troubleshooting.

The aim of this paper is to illustrate the Arrow-bot design and implementation for line following application based on real time system concept. This robot is called Arrow-bot since its PCB base was designed with arrow shape.

In section 2, a further detail on robot design is focused to describe the system architecture, hardware design. Then, the software design based on the real-time concept will be explained in section 3. Section 4 briefly discusses on hardware setup and real-time implementation result. Finally, section 5 concluded the overall of this paper.

2 Robot Design

Arrow-bot was designed based on criteria of a desktop size, low cost, and open source of hardware/software system; and motivated by previous research in [5], [7 – 9]. Robot size should be small and capable of moving

^a Corresponding author: mfauzi@uthm.edu.my

around on a table with 500 mm x 400 mm working space. The hardware selection and robot design had to use the minimum number of hardware and a through-hole package of electronic components. Therefore, the students can understand easily how the system work and spend less time for hardware troubleshooting. The power source for the robot system relied on +5V power bank battery that commonly used for handphone charging. With this, the use of power converter and charging system can be eliminated.

2.1 System Architecture

The block diagram of Arrow-bot's system architecture is shown in Figure 1. This system uses an Atmega328P microcontroller chip that contains Arduino UNO firmware for controlling the robot application. With Arduino UNO firmware, we can directly program it through universal asynchronous receiver/transmitter (UART) interface pins. This microcontroller has features of 32KB flash memory and 2KB SRAM, 16 digital input/output including six pulse width modulation pins and six analog input pins. The robot uses two DC motors to move around and it depends on three infrared analog sensors for following the line. An incremental encoder is attached each wheel of the motors for determining the robot speed. Another two additional components that related to user action are an LED indicator and a button. The button is used to change the state of the system and the LED indicator shows the running status of the system.

Figure 1. Block diagram of system architecture.

2.2 Hardware Design

The hardware components had been selected based on the availability and easy for future maintenance. There is no surface mount device involved. The robot was designed based on a differential wheeled mobile robot with one castor at front. The complete electronic schematic design is shown in Figure 2. The main controller circuit comprises an external 16MHz clock, +5V voltage reference for analog to digital converter (ADC) peripheral, user reset button, and programmer/debugger interface. An interface between controller and PC module is needed for programming and debugging activities which can be FTDI USB to UART converter module or Bluetooth Serial Link (Bluefruit EZ-Link) module.

Figure 2. Electronic circuit schematic design.

Three IR sensors (U3-U5) were identical and designed for detecting dark surface with a higher value than a white surface. A motor driver used L293D for controlling the speed and direction of two brushed DC motors. Two miniQ robot chassis encoders from dfrobot.com were used to measure motor speed and connected to interrupt pins of the microcontroller. An LED indicator used sourcing mode connection and a user button needed software-enabled internal pull-up resistor for detecting high state signal.

3 Real-Time Software Design

The scope of software design for real-time application for this Arrow-bot system is based on pre-emptive scheduling for independent tasks. In practical work, we should write a complete task of coding and determine the worst-case execution time (WCET) for it. The WCET can be measured by using timer/counters, logic analysers and oscilloscopes; then we can perform calculations and analysis of a system task priority scheduling [2].

In the line following application, three independent tasks were created which are suitable enough for giving example to student and the real-time embedded system concept is based on rate monotonic schedulability test algorithm. Task A handled line following operation. Task B determined the robot speed and task C blinked LED indicator to show the system is in running mode. These multiple tasks will be managed their execution by a real-time operating system (RTOS). There is a few suitable open source RTOS freely available such as FreeRTOS, Duinos, ChibiOS/RT, and RTuinOS. Among these, I picked ChibiOS/RT since it has a good example to jump start in Arduino-based AVR microcontroller.

3.1. ChibiOS/RT

ChibiOS/RT implements fixed priority pre-emptive scheduling algorithm. Each task has its own priority level and can be in running, ready and waiting state. In this implementation, the ChibiOS/RT version 3.0.3 library were used and it can be downloaded at <https://github.com/greiman/ChibiOS-Arduino>.

3.2 Worst-Case Execution Time

The worst-case execution time (WCET) is the maximum processor (CPU) time needed to execute a task. It should be determined before schedule its priority and analyze its schedulability. We can directly write a complete code of system operation based on ChibiOS/RT library in Arduino IDE. The essential coding for Task A is given in Figure 2. This task started read analog line sensor and changed to binary data whether HIGH or LOW state. HIGH state means detecting the line. Then, in the if-else statement robotMotion function would be called according to the sensors position. First and second arguments in robotMotion function are for controlling speed and direction of motor 1 (left) and the rest for motor 2 (right).

```
//ADC data to Binary data-----
if (analogRead(A3)<550) IRL = HIGH; else IRL = LOW;
if (analogRead(A2)<550) IRM = HIGH; else IRM = LOW;
if (analogRead(A1)<550) IRR = HIGH; else IRR = LOW;
//Robot Navigation-----
if (!IRL&&IRM&&!IRL) robotMotion(255,0,255,0);
else if (!IRL&&IRM&&IRR) robotMotion(255,0,120,0);
else if (!IRL&&!IRM&&IRR) robotMotion(255,0,0,0);
else if (IRL&&IRM&&!IRR) robotMotion(120,0,255,0);
else if (IRL&&!IRM&&IRR) robotMotion(0,0,255,0);
else if (analogRead(A2)>999) robotMotion(0,0,0,0);
//Rest: wait for next cycle-----
chThdSleepMilliseconds(50);
```

Figure 3. Line following code for Task A.

Task B was to determine the robot speed that based on total pulse counted from encoder interrupt every 1 second as given in Figure 4. It started by reading total pulse and calculate the wheel speed for each motor. With the 42-mm wheel size and 12 pulses per rotation, we can get 11 mm distance for one pulse. Then, robot speed was calculated based on average speed of both wheels. The result of the total pulse of each wheel and robot speed were sent to PC terminal.

```
//Read total pulse from interrupt of encoder
pSpeed1 = pulse_ENC1;
pulse_ENC1 = 0;
pSpeed2 = pulse_ENC2;;
pulse_ENC2 = 0;
//Convert pulse to mm/s unit
mlSpeed=11*pSpeed1; //1pulse=(pi*42mm)/12=11mm
mrSpeed=11*pSpeed2;
//Calculate robot speed
robotSpeed=(mlSpeed+mrSpeed)/2;
//Send data to PC terminal
Serial.print(pSpeed1); Serial.print("\t");
Serial.print(pSpeed2); Serial.print("\t");
Serial.print(robotSpeed);Serial.println("mm/s");
chThdSleepMilliseconds(1000);
```

Figure 4. Robot speed measurement code for Task B.

Task C blinked LED by turning on and off at a certain rate. After completing writing all tasks and configuration setting, three additional output pins at D10, D11 and D12 were used for capturing WCET of Task A, B and C respectively. These pins were connected to a logic analyzer. At this stage, the period of each task was ignored. The WCET signal was captured is shown in Figure 3.

Figure 5. Worst-case execution time of each task.

3.3 Rate Monotonic Schedulability

There are two formal tests for task schedulability in a mono-processor system that are a simple and full test of rate monotonic schedulability (RMS) [1 – 2]. The simple test of RMS is based on Liu and Layland [10] as in Eq. 1. All tasks in a system are schedulable if total CPU utilization is less than or equal threshold, where n is concurrent independent tasks, P_i is period and C_i is CPU time for $1 \leq i \leq n$.

$$\underbrace{\sum_{i=1}^n \frac{C_i}{P_i}}_{\text{Total CPU Utilization}} \leq \underbrace{n \left(2^{\frac{1}{n}} - 1 \right)}_{\text{Threshold}} \quad (1)$$

In the Arrow-bot application, each task period was proposed and the total CPU utilization is stated in Table 1. The percentage threshold for three tasks is 77.98%.

Table 1. A simple test of RMS analysis.

Task	P_i	C_i (WCET)	Utilization
A	50 ms	0.38 ms	0.76 %
B	1000 ms	0.64 ms	0.064 %
C	100 ms	0.02 ms	0.02 %
Total CPU Utilization			0.844 %

The tasks were schedulable because total CPU utilization is less than the threshold. To know how long a task's worst-case completion time is, we need to perform a full test of RMS analytical method as given in Figure 6. The worst-case completion time, S_{IF} for each task was calculated as shown in Table 2. All tasks were meet their datelines since $S_{IF} < P_i$.

Let T be the set of n tasks in a system. A task assigned with an integer i – a priority number – is denoted T_i and its period is P_i .

1. Assign the tasks in T each with a different priority number 1, 2, ..., n according to RMS, such that for any two tasks T_i and T_j , $i < j$ provided period $P_i < P_j$ (note: the convention used here is that the smaller the priority number assigned, the higher the task priority);
2. For each task T_i iteratively compute S_{i0} and $S_{i1}, \dots, S_{im}$ until $S_{i(m+1)} = S_{im}$ (call it S_{iF}), where
 - a. S_{i0} = is the completion time for task T_i if no task will preempt T_i during its execution;
 - b. $S_{i(x+1)} = R_i(S_{ix}), x \geq 0$, is the completion time for T_i if higher priority tasks will preempt it during its execution, where
 - i. $R_i(t) = C_i + \sum_{j=1}^{i-1} C_j \times \left\lceil \frac{t}{P_j} \right\rceil$ is the cumulative demand for processing in the duration t ;
 - ii. the ceiling function $\lceil \cdot \rceil$ for a number $a.b$ (e.g. 5.4, $a = 5$, $b = 4$) is defined by

$$\lceil a.b \rceil = \begin{cases} a + 1 & \text{if } b > 0, \\ a & \text{if } b = 0. \end{cases}$$

3. If there exists a value $S_{iF} < D_i$ (D_i is the deadline for task T_i), then T_i is schedulable with worst-case completion time S_{iF} .

Figure 6. A full schedulability test using RMS [2].

Table 2. A full schedulability test of RMS analysis.

Task	Priority (i)	Description	C_i (ms)	P_i (ms)	S_{iF} (ms)
A	1	Line Following Control	0.38	50	0.38
B	3	Robot Speed Measurement	0.69	1000	1.09
C	2	LED blinking	0.02	100	0.4

4 Results and Discussion

4.1. Hardware Setup

The robot was developed according to a double layer of PCB design as in Figure 7. Then, it was assembled with electronics components, a power-bank, two wheels, and a castor. The power bank was the main power source for the Arrow-bot which produced 5.23V and the rated current consumption for the system was 0.3A. The complete Arrow-bot system is shown in Figure 8 and successfully operated on working area with 17 mm white line size.

Figure 7. 3D view of electronic printed circuit board.

Figure 8. Complete Arrow-bot system.

Both motors could be operated at maximum speed with 100% duty cycle of PWM and produced 10 pulses of encoder signal in every second. The wheel diameter is 42 mm and one rotation produces 12 pulses. So, the maximum speed of robot could move was 0.11 m/s (50 RPM) and shown in the PC terminal as in Figure 9. Three IR sensors were located about 8 mm apart and 7 mm gap to the floor surface. The volume size of Arrow-bot was 9.2 mm length, 12 mm width and 4.7 mm height. The cost of components for one Arrow-bot system was RM 370.00.

Figure 9. Robot's speed measured by motor encoder.

4.2. Real-Time Application

A complete 144 lines of code based on RMS priority assignment in section 3.3 was tested successfully. The binary sketch size was 7808 bytes which used 24% of a 32,256 byte of internal microcontroller Flash memory and consumed 1004 bytes (49%) of RAM usage. The signal of the first cycle of each task was captured and Table 3 shows the measured parameters of task temporal scope. There was a time latency between context switching from Task A to Task B and Task B to Task C with spent 0.122 ms and 0.137 ms respectively.

Table 3. Task-scope for the first execution cycle signal.

Task	A	B	C
Start delay	0.000 ms	0.501 ms	1.270 ms
Elapse time	0.379 ms	0.632 ms	0.012 ms
Completion time	0.379 ms	1.133 ms	1.281 ms

The system had been successfully tested and the complete signal of a cycle task operation was captured as shown in Figure 10. The result of a cycle time was measured for determining the percentage error. As tabulated in Table 4, the period of every task was acceptable with low than 5% of error. The line following Arrow-bot application was considered real-time system since it had predictability and deterministic response time. By using RTOS in embedded microcontroller system for handling multiple tasks has made our life easier where we just focus on designing an individual task operation without bothering other tasks.

Table 4. Percentage error of period setting.

Task	i	Period, P_i		
		Target (ms)	Measured (ms)	% Error
A	1	50	48.78	2.05
B	3	1000	999.60	0.04
C	2	100	98.71	1.29

For future development, the Arrow-bot can be improved for smoothing and accurate navigation technique proposed by M. Engin and D. Engin [11]. In terms of task processing and scheduling, the Arduino UNO-based microcontroller has capable of handling more tasks but the designer should be careful on RAM usage. Therefore, we should select suitable data types and determine whether to use global or local variables.

5 Conclusion

This paper has presented the design and development of line following differential wheeled mobile robot. This robot is called Arrow-bot purposely designed as a teaching tool for real-time embedded system course at UTHM.

Figure 10. A single cycle of tasks operation with encoder.

The design consideration for this Arrow-bot was based on criteria of a desktop size, low cost, and open source of hardware/software system. Therefore, the Arduino UNO-based microcontroller and ChibiOS/RT real time operating system were chosen. The software coding for white line following application was designed and developed in Arduino IDE, then the scheduling task was relied on rate monotonic scheduling (RMS) procedure and was tested successfully. The Arrow-bot could be extended to other application or add another subsystem which creating more tasks but we must carefully choose suitable data types and location of variable declaration since the microcontroller has limited RAM size.

References

1. M. Chetto. Real-time Systems Scheduling, **1**, ISTE Ltd and John Wiley & Son, Inc (2014)
2. C. G. Leedham and K.-T. Seow. *Embedded real-time systems: introductory concepts and tool*, Pearson/Prentice Hall (2007).
3. R. George and Y. Kanayama. 1996. A rate-monotonic scheduler for the real-time control of autonomous robots. *IEEE International Conference on Robotics and Automation (ICRA)*, **3**, 2804–2809 (1996).
4. R. M. Harlan, D. B. Levine, and S. McClarigan. The Khepera Robot and the kRobot Class: A Platform for Introducing Robotics in the Undergraduate Curriculum. *Thirty-second SIGCSE Technical Symposium on Computer Science Education*, 105–109 (2001).
5. F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klapacz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. 2009. The e-puck, a Robot Designed for Education in Engineering. *9th Conference on Autonomous Robot System and Competition*, **1**, 59–65 (2009).
6. J. M. Gomez-de-Gabriel, A. Mandow, J. Fernandez-Lozano, and A. Garcia-Cerezo. Mobile Robot Lab Project to Introduce Engineering Students to Fault Diagnosis in Mechatronic Systems. *IEEE Transaction on Education*. **58**, 187–193 (2010).
7. I.W. Park and J.O. Kim. Philosophy and Strategy of Minimalism-based User Created Robots (UCRs) for Educational Robotics - Education, Technology and Business Viewpoint. *International Journal of Robots, Education and Art*, **1**, no. 1. 26–38 (2011).
8. R. Pérula-Martínez, J. M. García-Haro, C. Balaguer, and M. A. Salichs. Developing Educational Printable Robots to Motivate University Students Using Open Source Technologies. *Journal of Intelligent & Robotic System*, vol. 81, no. 1. 25–39 (2015).
9. F. M. López-Rodríguez and F. Cuesta. Andruino-A1: Low-Cost Educational Mobile Robot Based on Android and Arduino. *Journal of Intelligent & Robotic System*, **81**, no. 1. 63–76 (2015).
10. C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, **20**, no. 1. 46–61 (1973).
11. M. Engin and D. Engin. Path planning of line follower robot. *5th European DSP Education and Research Conference (EDERC)*. 1–5 (2012).